Tartu Ülikool
Arvutiteaduse instituut
Informaatika õppekava

Riko Piirisild

Bridži kaitsemängu analüüsi teostava treeningprogrammi loomine

Bakalaureusetöö (9 EAP)

Juhendajad: Ahti Põder, Kristo Väljako

Tartu 2024

Bridži kaitsemängu analüüsi teostava treeningprogrammi loomine

Lühikokkuvõte

Käesoleva bakalaureusetöö käigus luuakse kaardimängule bridž õppe- ja treeningprogramm, mille eesmärk on analüüsida kaitsemängu. Tutvustatakse bridži reegleid, antud probleemi olulisust tulemustele ning vastava programmi loomise protsessi ja tulevasi edasiarendamise võimalusi. Lõpptulemusena valmis programm, mis mängijale teadaoleva info kohaselt teostab analüüsi ning annab kasutajale tulemuseks teda huvitavate kaartide tõenäosusliku paiknemise.

Võtmesõnad

Bridž, kaardimängud, Java, programm, analüüs

CERCS

P175 Informaatika, süsteemiteooria

Creating a training program for analysing the defensive play in the card game bridge

Abstract

In this bachelor thesis, a training and teaching program for bridge was created to help players analyse the defensive play of the game. The thesis covers the rules of bridge, the importance of given problem and the creation of the program with further developing options. As the final result, the program will analyse the information known to the player and give the user the outcome which shows the probability of placement for the cards important to the player.

Keywords

Bridge, card games, Java, program, analysis

CERCS

P175 Informatics, systems theory

Sisukord

Sissejuhatus

Bridž on noorte seas aina enam populaarsust koguv mõttesport, mis arendab nii matemaatilist mõtlemist kui ka analüüsioskust ning omab ka seltskondlikku väärtust. Töö teemaks valiti just bridžiprogrammi loomine, sest autor on Eesti juunioride koondise bridžimängija ning õpetanud mängu ka algajatele. Soovisin bakalaureusetöö kirjutada enda jaoks olulisel teemal ja lahendada probleem, millega olen ise kokku puutunud. See kaardimäng on väga nüansirikas ning oskuslikuks mängijaks saamine nõuab palju aega, kuid tänu tehnoloogia arengule on võimalik kasutada treeningprogramme, mis kiirendavad mängija taseme arengut.

Töö eesmärk on kirjutada programm, mis teostab bridži kaitsemängu analüüsi ning annab mängijale tagasisidet matemaatilise tõenäosuse kujul. Bridžis on vaja mängijal teha otsuseid lähtuvalt talle teada olevast infost ning neid otsuseid tehakse enamasti varasema kogemuse põhjal. Küll aga on võimalik neid otsuseid teha matemaatiliselt korrektselt, kui mängija arvutab välja tõenäosuse tulemuse seisukohalt oluliste kaartide paiknemise kohta teistel mängijatel. Kuna analüüs on keeruline ning tõenäosust mõjutavaid faktoreid palju, siis oleks vaja luua programm, mis seda mängija eest teeb. Sellise programmi kasutamine mängu ajal on keelatud, kuid treeningprogrammina õppimiseks ja võistlusjärgseks analüüsiks mitte.

Bridži treeningprogramme on olemas küll mitmeid, kuid kuna mängul on tohutult nüansse, pole vastavat probleemi lahendavat programmi veel loodud või vähemasti mitte avalikult kättesaadavaks tehtud. Bridž koosneb suures plaanis kolmest osast: pakkumine, väljamäng ja kaitse. Esimese kahe kohta on loodud väga tõhusaid programme, millest tuleb lähemalt juttu teoreetilise tausta peatükis, kuid kaitsemängu jaoks on praegu vaid raamatud. Kaitsemäng on bridži kõige keerulisem osa, sest eeldab otsuste tegemist piiratud info põhjal ning samuti koostööd enda partneriga. Kuna kõrgel tasemega turniiridel on iga tulemus väga oluline ning võib muuta tulemust drastiliselt, on kaitsemängus tehtavad vead tihti väga suure kaaluga.

Käesolev töö koosneb kolmest peatükist: teoreetiline ülevaade, programmi koostamine ning programmi rakendamine ja testimine. Esimeses peatükis on välja toodud bridži reeglid ja mängu käik, mis on olulised mõistmaks programmi tööd, ning kasutatava märgistuse ja terminite selgitused. Samuti probleemi kirjeldus ja selgitus, milles üldse seisneb probleemi olulisus

mängule ning kuidas loodud programm probleemi lahendab. Teises osas on toodud välja programmi loomise protsessi kirjeldus koos selgitustega programmi iga osa kohta ning kirjeldatud rakendatud meetodeid toimiva lahenduse loomiseks. Viimases peatükis on toodud välja programmi testimine, selle kättesaadavaks tegemine ning analüüs potentsiaalseteks tulevasteks edasiarendusteks, mis on vajalikud programmi laiema üldsuseni viimiseks.

Esimese lisana on töö lõpus välja toodud maailma bridžiföderatsiooni ametlikud reeglid, millele põhineb teoreetiline taust ja loodud programmi raamistik, ning teise lisana ligipääs töö käigus valminud programmile ja selle koodile.

1. Teoreetiline ülevaade

Selles peatükis kirjeldatakse lähemalt, mis on bridž ja tutvustatakse selle terminoloogiat, selgitatakse, mis on selles mängus töö seisukohast kõige olulisemad arusaamad, miks just kaitsemäng oluline on ning millised programmid on juba olemas ning kuidas töö raames loodav rakendus lahendab probleemi, millele varasemad õppeprogrammide arendajad pole tähelepanu pööranud.

1.1 Kaardimäng bridž

1.1.1 Mängu kirjeldus

Maailma bridži föderatsiooni kohaselt on bridž kõrgetasemeline tihi võtmise kaardimäng, mis on mõeldud meeldivaks aja veetmiseks nelja inimese ja kaardipaki olemasolul.[1] Tegemist on rahvusvahelise olümpiakomitee poolt tunnustatud mõttespordiga ning seda loetakse kõige võistluslikumaks kaardimänguks kogu maailmas, sest hea tulemuse saavutamiseks on viidud õnne roll miinimumini.

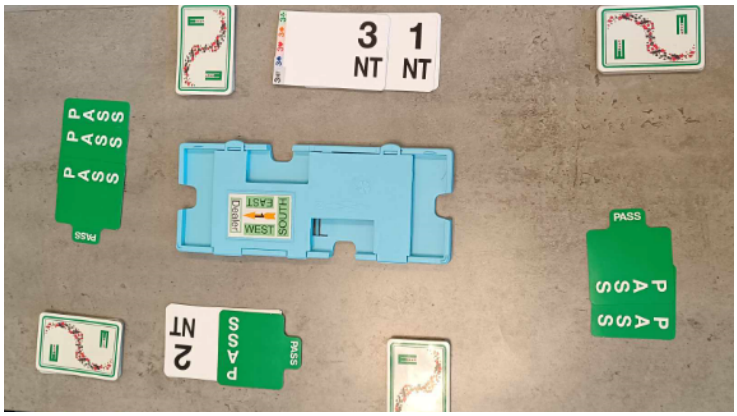
1.1.2 Mängu algus ja kaartide hindamine

Joonis 1, Bridži kaardid, Erakogu

Bridži alustalade mõistmiseks peab kõigepealt võtma mängu lahti väikesteks osadeks. Tegemist on nelja mängijaga mänguga, kus iga mängija istub ühte ilmakaarde - põhi (tähistus N), lõuna (tähistus S), ida (tähistus E) ja lääs (tähistus W) ning omavahel mängivad paaris põhi-lõuna ja ida-lääs. Viiskümmend kaks kaarti neljas mastis kahest ässani jagatakse võrdselt kõigi mängijate vahel, seega on igal mängijal käes kolmeteist kaarti. (Joonis 1) Enda kätt - 13 kaarti - hinnatakse kahe printsiibi alusel: kui mitu kaarti on igas mastis ehk mitu ristit, mitu ruutut, mitu ärtut ja mitu pada on vastavalt mängija käes ning seejärel loetakse kokku pildipunktid, mis saadakse liites kokku enda poisid (tähistus J), emandad (tähistus Q), kuningad (tähistus K) ja ässad (tähistus A) ning omistatakse neile punktid vastavalt 1, 2, 3 ja 4. See tähendab, et kokku on kaardipakis 40 punkti.

1.1.3 Pakkumine

Mängu esimest faasi nimetatakse pakkumisvõistluseks, kus kõik neli mängijat proovivad anda parimat võimalikku infot enda kaartide kohta, et leida sobiv leping oma partneriga kahe peale, mille eesmärk on fikseerida mänguks parim võimalik trump (mast, mis on tugevam kui kõik teised) ning tihide arv (kui mitu ringi, kus iga mängija käib ühe kaardi suudab pakkumise võitnud pool endale saada). Pakkumisel on kokku seitse korrust (mis tähistavad vastavalt 6+korrus valemi alusel, kui mitu tihi peaks võtma), kus igal korrusel on viis erinevat astet ning igal korrusel on võimalik kõrgemat astet madalama peale pakkuda. Tugevusjärjestus on siis risti (tähistus C), ruutu (tähistus D), ärtu (tähistus H), pada (tähistus S) ja trumbita (tähistus NT). Eesmärk on partneriga eelnevalt kokku lepitud süsteemi alusel (nn salakeel, mille alusel saab iga pakkumisega edastada mingi info enda käe kohta oma partnerile) jõuda kõige optimaalsemasse lõpplepingusse. Pakkumine lõpeb, kui on järjest pandud kolm passi lauale ehk on aktsepteeritud, et nüüd mängitakse viimasena pakutud lepingut. (Joonis 2)



Joonis 2, Pakkumise näide, Erakogu

1.1.4 Punktiarvestus

Pakkumise käigus on oluline jõuda kõige optimaalsemasse lepingusse, sest punktiarvestuses on erinevad boonused, mis annavad rohkem punkte kui tavatulemus. Punktiarvestus on hästi lahti seletatud Funbridge'i veebilehel [2], mis on virtuaalne mängukoht bridžihuvilistele. Tavatulemuse arvestus käib nii: iga 7. tihi annab sõltuvalt kokkulepitud lepingu mastist risti ja ruutu (odavmastide) puhul 20 punkti, ärtu ja paja (kallismastide) puhul 30 punkti ning trumbita mängimise puhul 40 punkti. Iga tihi pärast esimest seitset tihi annab sõltumata mastist 30 punkti. Teiste boonuste suurus sõltub aga suuresti tsoonsusest, kus kaardikarbi peal on iga jaotuse alguses kirjas ilmakaared, mis on kas rohelisel/valgel või punasel taustal, mis tähistavad vastavalt odavat ja kallist tsooni. Preemiaid saab bridžis geimide eest, mis on vastavalt pakutud 3NT, 4H, 4S, 5C ja 5D. Preemia suurus mitte tsoonis on 300 punkti ja tsoonis 500 punkti. Lisaks

on ka veel väikse-slämmi (pakutud leping 6 korrusel) ja suure-slämmi (pakutud leping 7 korrusel) boonus, mis on vastavalt 500 või 750 punkti ja 1000 või 1500 punkti. Selle olulisuse näiteks saab tuua lepingu tulemused kaks trumbita mängitud 10 tihile ehk võetud kaks ületihi ja kolm trumbita mängitud 10 tihile ehk võetud üks ületihi - mõlemal korral oli sama trump ja sama tihide arv, kuid esimese puhul saab 50 punkti lepingu eest, 40 punkti 7. tihi eest ja 30 punkti iga järgneva tihi eest ehk kokku 50 + 40 + 30 + 30 + 30 = 180 punkti, kuid 3nt+1 puhul lisandub ka geimipreemia vastavalt 300 punkti, seega arvutus tuleb 300 + 40 + 30 + 30 + 30 = 430 punkti. Seega +50 osamängu boonus (ei jõua geimi või slämmi) muutub +300 geimiboonuseks. See on aga tohutult suur ja oluline erinevus. (Joonis 3)

Joonis 3, Punktiarvestus, Erakogu

Viimaks on võimalik ka lepinguid kaotada, mis tähendab, et odavas tsoonis saavad vastased iga tihi eest, mis väljamängija (pakkumise võitnud mängija, kes otsustab mängu käigu üle (vt lähemalt järgmises peatükis)) vähem võtab kui lubatud, +50 punkti ning kallis tsoonis +100 punkti. Kuna bridžis saadakse punkte alati vastastikuliselt, siis arvestakse punkte alati väljamängija perspektiivist kogu töö vältel ehk väljamängija saab iga alttihi eest -50 või -100 sõltuvalt tsoonsusest.

Lisaks on pakkumises olemas selline kaart nagu kontra, mida saab suvalisele vastase pakkumisele peale panna, mis üldjoontes viitab mängimist kõrgematele punktidele ja veendumust, et seda lepingut väljamängija võita ei suuda. Väljamängivast liinist võivad mõlemad mängijad selle peale omakorda lisada rekontra, mis suurendab punkte veelgi nii juhul kui leping võidetakse, kui ka siis kui leping taha läheb. Mõlemal kaardil võib sõltuvalt olukorrast olla erinev tähendus, kuid punktiarvestuse perspektiivist võib seda mõista selliselt. Töö seisukohalt ei ole aga nende punktide välja toomine oluline, sest keskendutakse optimaalsele mängutehnikale sõltumata punktide arvust.

1.1.5 Kaardimäng

Kui pakkumine on lõpule viidud, saab pakkumise võitnud paarist väljamängijaks see mängija, kes pakkus vastavat masti esimesena. Näiteks kui lepinguks jäi 4S, mille pakkus mängija E, aga tema partner W pakkus varasemas pakkumisvõistluses 1S, saab väljamängijaks W. Tema vasakul

käel asuv mängija saab sooritada avakäigu ehk valida oma 13 kaardist ühe kaardi, mis asetada lauale, teades kõigi teiste mängijate kaartide kohta ainult pakkumisest selgunud infot. Kuna pakkumise jooksul vahetatav info on kõigile avalik, teavad kõik mängijad sama infot.

Kui avakäik on tehtud, saab väljamängija partnerist laud ehk asetab ta kõik oma 13 kaarti mastide järjestuses lauale kõigile nähtavale ning tema roll on käia alati täpselt see kaart, mida väljamängija ütleb ehk temal selles jaotuses enam otsustusõigust ei ole ning kõik mängijad teavad alati täpselt pooli kaarte ja nende asukohta. (Joonis 4) Nüüd algab mäng, kus igal 13 ringil peavad kõik neli mängijat asetama lauale kaardi ning kõige kõrgema kaardi omanik saab tihi endale ning kaart pööratakse lauale tagurpidi. Sealjuures on oluline, et tihi saab endale liin ehk kui E võtab tihi, siis see on neil W-ga kahepeale. Küll aga järgmise kaardi tihisse käib suurima kaardi pannud mängija. Lisareeglina on oluline märkida, et iga mängija, kellel on käes vähemalt üks kaart esimesena tihisse asetatud kaardimastiga samast mastist, peab sellest mastist kaardi käima. Juhul kui mast on otsa lõppenud, võib mängija asetada suvalise kaardi, aga see läheb tihi arvestusse ainult juhul, kui see on trump. Näiteks lepingu 2S puhul, kui avakäigu tegija asetab lauale ruutu 3, lauast asetatakse ruutu K ja kui partneril ei ole ruutus ühtegi kaarti, saab ta käia suvalisest mastist kaardi - ärtu ässa käimine tihi ei mõjuta kuigi see on suur kaart, sest see pole ei väljakäidud mast ega ka trump, see-eest aga pada 3 käimine muudab automaatselt selle kaardi selleks hetkeks kõige suuremaks tihisse käidud kaardiks, sest trump on tugevam kui väljakäidud mast.

Joonis 4, Mängu seis peale avakäiku, Erakogu

1.1.6 Kaitsemängu olulisus

Kogu eelneva kokkuvõtteks saab selgeks tõsiasi, et nii pakkumise kui väljamängu saab ära optimeerida üsna lihtsalt - lepingu väljamängivat poolt kontrollib üks mängija, kes teeb otsuseid nii enda kui partneri kaartide eest ning seeläbi saab välja mõelda parima matemaatilise plaani üksi, ning pakkumises on võimalik luua kokkulepe iga võimaliku pakkumusjärgnevuse kohta ja

seejärel see lihtsalt pähe õppida (teoorias saavutatav, kuid praktikas väga ajamahukas). Kaitsev pool aga peab pimesi lähtuvalt osalisest infost ja kokkulepitud signaalidest (mõne kaardi käik võib anda lisainfot kõigile lauas olijatele aga eelkõige partnerile) suutma käia perfektseid kaarte, et piirata väljamängija nii vähestele tihidele, kui vähegi võimalik.

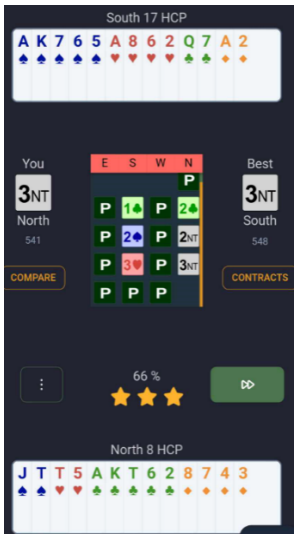
Üks bridžimaailmas enim hinnatud mängija ja õpperaamatute autor Edwin B. Kantar andis oma raamatus "Defensive bridge play" [3:3] edasi kõige olulisema mõtte, mida on parafraseerides öelnud ka teised mängu suurmeistrid, näiteks Bogdan Vulcan [4] ja David Bird [5]: "Ma olen valmis väitma, et suure tõenäosusega antakse rohkem tihisid ära valesti mängimise tõttu, kui pakkumises ja väljamängus kokku. Keskmine mängija ei saa isegi aru, kui palju ta kaitses tihisid ära annab ebaselgete signaalide, äravisete, õnnetute avakäikude ja kaartide mittelugemise tõttu".

Kuna pakkumises on kogu info avalik ja väljamängu puhul on vastutav ainult väljamängija mõlema poole eest, siis eksida on raskem ning kõrgel tasemel tuleb seal vigu ette harva. Küll aga võib hea kaitse vs. halb kaitse muuta kõike. Alles hiljuti otsustati 2023. aasta Eesti paaride meister bridžis viimases voorus ning teisele kohale jäänud paar oleks turniiri võitnud, kui otsustavas jaotuses oleks kaitses üks tihi vähem ära antud.

1.2 Programmi funktsionaalsus

1.2.1 Olemasolev tarkvara

Bridžimaailmas on viimasel ajal tõusujoones noorema generatsiooni pealekasv, mistõttu on hea, kui on olemas abistavaid programme, mis kiirendavad oskuste paranemist ka väljaspool füüsilist bridžilauda. Hetkel on neid programme sisuliselt kolme sorti: pakkumine, bridžimäng ja analüüs.



Joonis 5, Kuvatõmmis Cuebidsist

Pakkumise jaoks on kõige populaarsem programm Cuebids [6], kus on võimalik harjutada ainult pakkumist partneriga ning seeläbi arendada enda süsteemi ja üksteisest arusaamist. See on hea viis pakkuda selliselt, et vastasteks on robotid, mitut jaotust samaaegselt ning avada programm pakkumiseks endale sobival ajal mõlemal partneril. (Joonis 5)

Bridžimängu harjutamiseks on kõige populaarsem BridgeBase [7] ning populaarsust koguv Funbridge [2]. Nendes programmides saab virtuaalselt istuda ühe laua taga ükskõik milliste teiste

mängijatega üle kogu maailma ning harjutada kogu mängu tervikuna, kuid sealse mängu põhjal saab analüüsi teha ainult pakkumisvõistlusele, kaitse- ja väljamängule mitte.
Viimaks on olemas Dealmaster [8] analüüsimiseks, mis on töö kirjutamise hetkel ainus kommertslikult kättesaadav kaardimängu analüüsiv programm bridžimaailmas. Selle abil on võimalik analüüsida, mis oleks mingis olukorras parim võimalik pakkumine lähtudes teadaolevast infost. Lisaks on võimalik saada tõenäosus, mis on parim võimalik avakäik mingis kindlas olukorras. Aga mida see programm ei tee, on analüüs jaotuse mängimise ajal.
Seda probleemi lahendabki töö käigus valmiv programm, mis suudab igal mängu hetkel arvutada välja kaitsva poole jaoks oluliste kaartide asukohtade tõenäosust ning seeläbi annab parima võimaluse õppida, areneda ja oma mängu analüüsida nii algajatel kui ka edasijõudnutel.

1.2.2 Probleemi lahendamine programmi abil

Hea kaitse puhul võib mõnikord tekkida olukord, kus kõik ei ole nii selge ja peab mängima hoopis matemaatilisele tõenäosusele. Näiteks võtab E risti ässaga tihi ja vaatab enda käes olevat viimast kahte kaarti, milleks on väike ärtu ja väike pada. Nüüd näeb ta, et lauas on samuti väike ärtu ja väike pada ning teab varasemaid kaarte jälgides, et väljamängijal peab olema ruutu äss ning üks teadmata kaart ja partneril kaks teadmata kaarti. Kolme teadmata kaardi seas on veel ka paja A ja ärtu A ning veel üks väike ärtu. Ja E teab, et partneril mõlemat ässa ei ole, seega peab ta välja mõtlema, et kumb on partneril, sest muidu võtab vastane viimased kaks tihi ära.
Kui on pikad turniirid ning selle jooksul mängitakse sadu jaotusi, näiteks 5 päeva jooksul kokku 250 jaotust - nagu mängis 2023. suvel Eesti juuniorkoondis maailmameistrivõistlustel - ei pruugi see mõttekäik enam alati õige lahenduseni viia. Küll aga on välja toodud, et just sellistest jaotustes pikkadel turniiridel eraldatakse head mängijad väga headest mängijatest.
Loodava programmi eesmärk on anda kasutajale võimalus lisada analüüsimiseks kogu talle teadaolev info igal mängu hetkel ning saada vastuseks protsentuaalne tõenäosus, kelle käes on tõenäolisemalt lepingu perspektiivist oluline kaart või olulised kaardid. Kuigi sellise programmi kasutamine on turniiride ajal keelatud, annab see turniiril tehtud vigu hiljem analüüsides võimaluse hinnata, kas kasutaja tehtud otsus oli lihtsalt õnnetu või oli matemaatiliselt võimalik teha parem otsus. Sellest lähtuvalt saab mängija korrigeerida oma signaale partneriga, et paremini edasi anda mõistetavat ja üheselt arusaadavat infot, või hoopis paremini mõista, mida

eksinud pool peaks tulevikus teisiti tegema, et mitte olla sellises olukorras, kus on võimalus eksida või peab pimesi arvama.

1.2.3 Programmi rakendamine

Loodud programm on kasutamiseks bridžiõppel, mis on peamiselt suunatud juunioritele ehk alla 26 aastastele mängijatele. Kõige raskem osa noorte õpetamisel on anda neile edasi õpetussõnad, kuidas rasketes mänguolukordades teha korrektseid otsuseid. Noorte kogemus on pea alati õpetaja omast väiksem ning seega ei ole ka nende n-ö sisetunne nii treenitud, et teha õigeid otsuseid tunnetuse pealt. Seda tunnetust saab aga valminud programmi abil treenida, sest see annab matemaatilise kinnituse õigele plaanile ning seeläbi saab õppija ka analüüsida, millele peaks järgmisel korral sarnases olukorras mõtlema ning mis on sel juhul kõige optimaalsem mänguplaan.

Programmi saab kasutada nii treeningutel kui ka turniiri ajal, kui on voorude vahel pausid või kui mängupäeva lõpus päeva jooksul juhtunut analüüsitakse. Kui varasemalt pidi õppimisel usaldama endast kogenumate ütlusi või lihtsalt mängima suures koguses jaotusi, et õppida ära tundma erinevaid olukordi, ning katsetamise abil leidma õige plaani, siis nüüd saab seda protsessi kiirendada selle programmi abil.

2. Programmi koostamine

Järgnevas peatükis kirjeldab programmi koostamise protsessi ning analüüsitakse erinevaid etappe ja nendega kaasnenud probleeme. Kuna bridž on nüansirikas ja keerukas kaardimäng, kujunes programmi koostamine oodatust raskemaks. Iga etapi juures on toodud välja programmiosa eesmärk, koostamise protsessi kirjeldus, tekkinud probleemid ning kuidas need lahendati.

2.1 Programmeerimiskeele valik

Valisin programmeerimiskeeleks Javascripti mitmel põhjusel. Eelkõige on mul selle keelega kõige rohkem kogemusi nii ülikoolist kui ka eraelulisel tasemel. Samuti meeldib mulle väga IntelliJ Idea kasutajaliides Javascripti koodi koostamisel, sest see teeb vähese kogemusega programmeerija töö palju lihtsamaks tänu lihtsale ja loogilisele kasutajaliidesele, mille abil on koodi mugavam koostada ning funktsionaalsust liideste kaudu ilma suuremate probleemideta lisada. Oluliseks aspektiks oli ka ette planeerimine, kuna programm peab viimaks olema selline, et kasutaja saab selle käivitada ja kohe kasutama hakata. Olen ka varem ülikoolis tudengiprojekti raames JavaFX abil koostanud rakenduse, seega tundus kõige loogilisem kasutada just seda. Küsisin ka tuttavate programmeerijate käest ning nemad tõid ühe valikuna välja ka Pythoni, kuid kinnitasid, et parim on valida keel, milles end kõige mugavamalt tunnen.

2.2 Plaani koostamine

Esimese asjana asusin koostama plaani, et kuidas kogu see programm lõpuks välja võiks näha ning mis funktsionaalsust omada. Keeruline oli leida head ideed, millest alustada, aga otsustasin alustada tagurpidi ehk mida on vaja, et programm ütleks mängijale, kus asub tulemuse jaoks oluline kaart. Selleks peab programm teadma mängijate kaarte. Selleks, et teada mängijate kaarte, on vaja teada, mis kaardid üldse võivad mängijal olla. Selle jaoks peab programm teadma eeldusi punktide ja mastipikkuste kohta, aga enne seda peab programm mõistma, mis on üldse punktid ja mastipikkused. Enne seda peab programm mõistma, millised on bridži reeglid, et kaarte korrektselt jagada. Seega alustasin kaardipakist ning sellest, et programm mõistaks kaartide väärtuseid ning seejärel asusin ehitama igat järgnevat osa programmist.

2.3 Programmi osad

2.3.1 Kaartide jagamine ja bridži reeglid

Kaartide jagamiseks oli tarvis esmalt luua kaardipakk ning neli mängijat, kellele neid kaarte jagatakse. Kaardipaki loomisel kasutasin massiivi, kus iga element on üks 52 kaardist ning samuti lõin igale mängijale oma massiivi, mis sisaldab tema käes olevaid kaarte. Selleks, et iga jaotus oleks unikaalne, proovisin esmalt valida suvaliselt ühe mängija, kellele jagada kaardipakist esimene kaart ja siis teine kaart jällegi suvalisele mängijale ja nii seni, kuni kaardid otsa saavad.

Siin tekkis aga kaks olulist probleemi: mõnel mängijal oli mitu sama kaarti ning mõnel mängijal oli rohkem kaarte kui 13. Selle tingis olukord, kus programm kontrollis ainult seda, kas kõik 52 kaarti on jagatud ega arvestanud reeglitega. Samuti ei olnud efektiivne lahendus iga kaardi jaoks leida suvaline mängija, kellele see jagada. Leidsin, et parim lahendus on luua järjestatud kaardipakist koopia ja segada see kaardipakk Java Utilitiesi käsu *shuffle* abil, mis annab palju parema suvalise järjestuse kui näiteks käsu *math.random* kasutamine. Seejärel asusin kaarte ühekaupa järjest jagama igale mängijale ning eemaldama seejärel kaardipakist jagatud kaardi. Nüüd ei saa tekkida olukorda, kus kaarte on topelt või mõnel mängijal on rohkem kui 13 kaarti.

2.3.2 Tulemuste väljastamine ekraanile

Tulemuste väljastamine näis kohe olevat probleem, sest selleks, et tulemusi kiirelt kontrollida ja informaatikakaugele inimesele loetavaks teha, pidin välja mõtlema mingi parema süsteemi kui massiivide tekstina väljastamise. Leidsin, et parim viis selleks on luua alamklass *Card*, mille eesmärk on muuta massiivide sisu loetavaks ning tekitada ka olukord tulevikuks, kus igale kaardile on omastatud kaks väärtust - mast ja sümbol. Seejärel sain väljastada tulemused mugavalt, sest igal kaardil oli olemas parameetrid “mast” ja “sümbol” ning pidin lisama ainult sõna “of”, et see oleks universaalselt loetav ja mõistetav. Selle loomise järel muutsin ka algset kaardipaki massiivi, millest sai nüüd kaks massiivi: üks kaardimastide jaoks ja üks kaardi sümbolite jaoks. (Joonis 6)

```
North
HCP: 10
Spades: T 5
Hearts: 7 4 3
Diamonds: K J T
Clubs: A Q T 9 4
South
HCP: 10
Spades: A K 7 6 4
Hearts: Q J 9 8
Diamonds: 9 6
Clubs: 8 6
East
HCP: 10
Spades: J 9 8 2
Hearts: K 6
Diamonds: Q 5 3
Clubs: K J 5 3
West
HCP: 10
Spades: Q 3
Hearts: A T 5 2
Diamonds: A 8 7 4 2
Clubs: 7 2
```

Joonis 6, Kuvatõmmis ekraaniväljundist

2.3.3 Pildipunktide ja mastipikkuste eeldused

Pildipunktide parameetri lisamine tundus esmalt tohutult keeruline ülesanne. Kaartidel on juba omadustena olemas mast ja sümbol, kuid nüüd tuleb veel lisada igale sümbolile ka numbriline väärtus, mis näitab, kui mitu pildipunkti vastav kaart väärt on. Leidsin, et hea lahendus on kasutada *Map* funktsioon, kus sain kaardi sümbolit tähistava tähe (ingliskeelsete lühendite kohaselt äss on A, kuningas K, emand Q ja sõdur J) siduda numbrilise väärtusega vastavalt 4, 3, 2, 1 ning anda kõigile ülejäänud kaartidele väärtuseks 0. Seejärel proovisin anda programmile ka teadmise, et kaardipakis on kokku alati maksimaalselt 40 punkti, kuid leidsin hiljem, et see ei ole tarvilik.

Seejärel oli tarvis programmile ette anda, kui palju pildipunkte iga mängija käes olla saab ning teha kindlaks, et programm jagab kaarte vastavalt eeldustele. Selleks leidsin inspiratsiooni esimest korda ülikoolis Java kasutamist õppides nähtud ülesandest, kus anti ette erineva väärtusega mündid ning pidime seejärel koostama programmi, mis kontrollib, kas nende müntidega on võimalik täpselt tasuda arvel olev summa või mitte. Seega lisasin kaartide jagamise funktsiooni, mida programm läbib rekursiooni abil, milles kontrollitakse enne iga järgmise kaardi jagamist, kas selle kaardi lisamisel mängija kätte ületab pildipunktide summa lubatud punktiarvu või mitte. Programm seejärel vastavalt kas lisab kaardi mängija kätte või võtab pakist järgmise kaardi. Muutsin ka funktsionaalsust vastavalt, et esmalt jagatakse ühele mängijale kõik tema kaardid ja alles siis võtab programm järgmise mängija.

See aga tekitas olukorra, kus programm lõpetas funktsioneerimise, kui kaarte jagada ei õnnestunud ning jagas tihti mängijale vähem kui 13 kaarti. Selle vältimiseks lisasin juurde kaks kontroll-lauset, kus esmalt programm kontrollib, kas kõigi mängijate pildipunktide eelduste summa on 40, ning jagab kaarte ainult siis, kui on. Seejärel kontrollib pärast iga kaardi jagamist, kas mängijal on juba 13 kaarti - kui on, siis läheb järgmise mängija juurde, ning kui mitte, siis proovib jagada järgmisi kaarte seni, kuni õnnestub.

Samuti lisasin võimaluse anda ette iga mängija minimaalne ja maksimaalne võimalik pildipunktide arv, sest bridžis on väga harva teada mõne mängija täpne punktide arv. Kuna see tegi aga programmi tohutult aeglaseks, sest programm jagas suvaliselt kaarte seni kuni leidis jaotuse, mis sobib eeldustega, tekitasin lahendusena enne kaardijagamise koodi käsu *Math.random* abil funktsiooni, mis valib iga mängija pildipunktide vahemikust suvalise väärtuse,

kontrollib seejärel, kas väärtuste summa on 40 - vajadusel valib uued väärtused - ning asub siis järgmise osa juurde.

Mastipikkuste etteandmine programmile kujunes aga keerukamaks ülesandeks. Otsustasin esmalt luua eraldiseisva koodi, mis kasutab samu sisulisi aspekte, kuid jätab välja pildipunktidega seonduva. Proovisin seda lahendada mitmel viisil, kuid ükski ei toiminud tervikult ega piisava kiirusega. Sobilikuks osutus lõpuks järgmine lahendus: lõin eraldi massiivi, mis sisaldab iga mängija jaoks sobilikke mastipikkuseid. Seega sain nüüd ette anda kaardijagamise funktsioonile kontrollmehhanismi, kus kaart jagatakse mängijale ainult juhul, kui vastava mastiga kaarti veel tohib mängijale jagada. Lubatud mastipikkuste massiivist tegin samuti koopia, kus iga kord, kui ühest mastist kaart ära jagati, lahutati üks selle masti pikkusest ehk jäi alles number, mis ütles, kui mitu selle masti kaarti on veel võimalik mängijale jagada. Teadsin juba pildipunktide kontrollimiseks loodud koodist, et pean lisama ka kontrolli, kas mastipikkuste eeldused vastavad reeglitele, et kokku ei ole kunagi rohkem kui 52 kaarti ja rohkem kui 13 kaarti igast mastist, ning lasta programmil proovida kaarte jagada seni, kuni jagatud kaardid vastavad kõigile reeglitele ja eeldustele.

Lõpuks oli vaja ühendada mõlemad funktsionaalsused ühtseks programmiks. Kuna struktuuriliselt olid mõlemad koodid sarnased, oli ainult vaja olla tähelepanelik, et kõik toimuks õiges järjekorras: kontrollime, kas sellised punktid ja mastipikkused vastavad reeglitele; seejärel kontrollime, kas järgmist kaarti saame mängijale jagada, et see ei satuks vastuollu eeldustega ,ning siis jagame kaardi, viimaks kontrollime kõige lõpus enne tulemuste kuvamist, kas iga mängija käes olevad kaardid vastavad etteantud eeldustele ja bridži reeglitele.

2.3.4 Teadaolevate kaartidega arvestamine

Kuna programm peab olema suuteline lõpuks igal mängu hetkel õige vastuse andma, oli vaja lisada programmile funktsionaalsus, kus iga mängija kohta on mingid kaardid juba teada. Kuna bridži kaardimängu osa ajal teab iga mängija alati vähemalt poolte kaartide asukohta, peab seda teadma ka programm. Selleks andsin programmile ette nö teise jaotuse enne muid tegevusi, kus on kirjas teadaolevad kaardid iga mängija kohta. Seejärel jagab programm need kaardid ka enda süsteemis vastavatele mängijatele ning eemaldab need kaardid kaardipakist, kust jagatakse kõik teadmata kaardid. Selleks, et programmi eelnev funktsionaalsus säiliks, oli vaja ära muuta

kontroll-laused, kus maksimaalne punktide arv oli nüüd teadaolevate punktide võrra väiksem ja mastide pikkused sisaldavad nüüd mitte masti kogupikkust, vaid lisatavate kaartide arvu.

2.3.5 Kasutajalt sisendi küsimine

Programm jagab nüüd kaarte edukalt kindlate sisendite puhul, kuid lõpuks on sisend vaja küsida kasutajalt, mitte koodis ette anda. Kõige lihtsam lahendus oleks olnud lasta kasutajal sisestada info täpselt selles formaadis, nagu see on koodis endas sees juba olemas, kuid programmi jaoks loogiline kirjaviis ei ole selles olukorras mitte mingilgi viisil lihtne, loogiline ega mugav kasutaja jaoks. Seega on vaja kasutajalt küsida tema jaoks kõige naturaalsemal viisil eelduseid: pildipunktid kujul miinimum-maksimum (näiteks 11-16), mastipikkused kujul padaärturuuturisti (näiteks 5431), kus esimene number tähistab pajakaartide arvu, teine number ärtude arvu, kolmas ruutude ja neljas ristide arvu ning teadaolevad kaardid kujul mastsümbol (näiteks S9), kus kahest tähemärgist koosneva sisendi esimene märk tähistab masti ja teine väärtust. Leidsin, et parim viis on küsida kasutajalt iga mängija kohta täpselt need kolm küsimust: Mis kaardid on selle mängija käest juba teada, mitu pildipunkti mängijal veel olla võib ja mitu kaarti tal veel mastides järel on. (Joonis 7)

```
Enter HCP range for North (min and max): 9 11
Enter HCP range for South (min and max): 9 11
Enter HCP range for East (min and max): 9 11
Enter HCP range for West (min and max): 9 11
```

Joonis 7, Kuvatõmmis ekraaniväljundist 2

Nüüd tuli asuda tegelema tõlkimisega bridži keelest programmikeelde. Teadaolevate kaartide sisendi puhul pidin pöörama tagurpidi juba varasemalt loodud funktsiooni, mis väljastas ekraanile loetaval kujul programmi poolt arvutatud info. Kasutades sõnetöötlust ja mitmeid tingimuslauseid, sain sisendist kätte kõik kaardid programmile mõistetaval kujul. Pildipunktidega oli asi lihtsam - tee sisestatud sõne sidekriipsu juurest pooleks (*split*) ning tekkinud massiivi esimese elemendi väärtus omistada miinimumiks ja teine element maksimumiks. Sarnaselt sai lahendatud mastipikkuste sõne pulkadeks lahti võtmine käsu *CharAt* abil ning sain need väärtused üsna kerge vaevaga lisada eelnevalt loodud allesjäänud mastipikkuseid tähistavasse massiivi.

2.3.6 Oluliste kaartide asukohtade tõenäosus

Programm suutis selleks hetkeks väga kiiresti leida ühe sobiva jaotuse, mis vastab kõigile eeldustele. Lõpuks peab aga programm andma - arvestades etteantud eeldusi - kasutajale mängija

jaoks oluliste kaartide tõenäosusliku asukoha. Selleks oli vaja programmi muuta väga mitmel viisil ning see võttis väga suure osa kogu programmeerimisele kulunud ajast.

Alustasin sellest, et programm võiks genereerida näiteks 100 jaotust vastavate eeldustega ning need kõik ekraanile väljastada, et saaksin igaühe sisulist õigsust vajadusel kontrollida lisaks ka oma teadmiste abil. See aga tekitas kohe probleemi, kus kaarte jagatakse hetkel ju pakist, kust see pärast jagamist eemaldatakse ning järgmiste jaotuste jaoks ei ole programmi jaoks kaardipakki, kust neid jagada. Seega oli vaja luua tsükliväline kaardipakk ning iga tsükli alguses teha vastavast kaardipakist koopia, millest siis programm kaarte eemaldama hakkas. Sarnaselt pidin ka salvestama ja kopeerima kõik muud massiivid, mida töödeldi või muudeti programmi tegutsemise käigus.

Nüüd oli vaja lisada loendur, mis paneb iga genereeritud jaotuse puhul kirja, kus asub mängija jaoks oluline kaart. Selleks lõin veel ühe alamklassi, kuhu lisasin neli int-muutujat. Iga sobiliku genereeritud jaotuse lõpus käib programm läbi kõigi mängijate käes olevad kaardid ning liidab ühe vastavale muutujale ning pärast 100 jaotust väljastab muutujad ekraanile. Oma mängukogemuse põhjal ja kasutajakogemust silmas pidades otsustasin, et igal mängu hetkel ei ole mängijal reaalselt rohkem kui kolm kaarti, mille asukoht talle oluline on või saab ta 3 kaardi asukoha abil järeldada teiste oluliste kaartide asukohad. Seega muutsin int-muutujad hoopis massiivideks ning asusin kirjutama tingimuslauseid, mida lõpuks sai väga palju. Kuna soovisin väljastada kasutajale kolme kaardi puhul kõikvõimalike kaardikombinatsioonide asukohtade tõenäosused, oli vaja nii õigete numbrite arvutamiseks kui ka väljastamiseks kontrollida iga jaotuse järel kõigi kombinatsioonide eksisteerimist ja vastavad väärtused lisada loendurisse ning hiljem ka see info loetaval kujul kasutajale edastada.

Nüüd tuli asuda programmi optimeerima, sest 100 jaotust ei ole piisav, et anda matemaatiliselt piisavalt korrektne tõenäosus. Seega on vaja genereerida palju rohkem jaotusi, kuid programmi töö ei tohiks võtta ka rohkem kui mõne sekundi, sest kui programm genereeriks tulemusi pikemalt, siis poleks selle kasutamine mugav ei turniiridel ega treeningutel. Eriti kui programm peab ühel hetkel olema kasutatav igas nutiseadmes. Kasutades programmi ajalise keerukuse kohta õpitud teadmisi ja pärast palju testimist leidsin, et optimiseeritud koodi puhul on kõige parema aja ja täpsuse suhtega genereeritud jaotuste arv 10 000, sest sellisel juhul annab programm piisavalt täpse tõenäosuse, aga ei kuluta samal ajal rohkem kui 10 sekundit tulemuse edastamiseks kasutajale. (vt peatükk 3.2) Kuigi tõenäosus muutub matemaatiliselt päriselt

tõeliselt täpseks alles väga palju suuremate arvude puhul, on bridži perspektiivist on oluline teada suurusjärku, kus näiteks tulemus 70% on risti kuningas Westi käes ja 30% Northi käes ei erine mängija jaoks sisuliselt 72% Westi ja 28% Northi tõenäosusest. Küll aga on 10 000 piisav arv, et spetsiifiliste eelduste puhul anda üsna täpne vastus ning väga laiade eelduste puhul anda piisava täpsusega vastus.

2.4 Programmi teisaldamine ühtseks rakenduseks

2.4.1 Sisulise osa teisendus

Nüüd on valmis programm, millel on olemas kõik bakalaureusetöö raames oluline sisuline funktsionaalsus. Edasi on vaja muuta Java abil käivitatav programm koos konsooliga rakenduseks, mida on võimalik ühe kliki abil igas Windows-seadmes ning kasutama asuda. Selle jaoks kasutasin JavaFX-i abi, et luua ja asuda kasutama jar-faili. Selle loomine oli IntelliJ puhul üsna lihtne, sest programmi oli sisseehitatud õpetus selle tegemiseks.

Selleks hetkeks oli olemas käivitatav programm, kuid see oli täiesti tühi. Nüüd oli vaja asuda ehitama programmi kasutajaliidest. Selleks lõin eraldi klassi, millesse asuda lisama sisulist funktsionaalsust. Kasutajalt küsitavad küsimused oli lihtne lisada: selleks pidin looma tekstiväljad ja kopeerima sinna varasemalt konsoolis esitatud küsimused. Iga küsimuse alla lisasin ka sisendkasti, millesse kasutaja lisatav tekst loetakse sõnena programmi sisse, mis oli keerulisem kui lihtsalt käskude muutmine, sest ma ei saanud enam kasutada *Scannerit*, vaid abiks tuli võtta *input*. Küll aga said sellega seonduvad probleemid lahendatud lihtsalt testimisega, kus muutsin sisselugemise ja töötlemise parameetreid seni kuni sain kätte samad andmed, mis varem konsoolist infot sisse lugedes. Vaja oli ka lisada nupp genereerimiseks, mis käivitab programmi funktsionaalsuse.

Selle kõige tagajärjel tekkis nüüd kaks probleemi: esiteks ei tulnud programm toime kõikvõimalike kasutaja sisendis sisalduvate vigadega ning programm jooksis kokku ning teiseks ei väljastatud ekraanile tulemusi. Teise probleemi lahendus oli lihtne - selleks oli vaja lisada uus tekstiväli, kuhu väljastati kõik tulemused sarnases formaadis nagu varem konsoolis. Probleemide tuvastamise jaoks oli vaja programmi töö peatada ning kuvada probleem kasutajale väga mitmes olukorras: kasutaja jättis mõne tekstivälja tühjaks; kasutaja lisas rohkem kui kolm olulist kaarti, mille asukohti programm arvutab; kasutaja lisatud punktivahemikud ja mastipikkused ei ole

kooskõlas bridži reeglitega; kasutaja on andnud punktivahemikud või mastipikkused väärtusega, mis ei ole 0 mängijatele, kelle kõik kaardid on juba teada; kasutaja sisestab vale kaardi või sama kaarti mitmel korral. Selleks lisasin koodi enne iga funktsiooni rakendamist kontroll-lause, mis väljastab tulemusena sõne, mis tuvastatakse käivitamise klassis ning vastav tõrge kuvatakse kasutajale ja peatatakse programmi töö. Kõik vähegi loogilised tõrked kasutaja sisendis said läbi vaadatud ning nüüd oli olemas käivitatav programmiaken, mis omab korrektset funktsionaalsust.
Viimaks oli vaja lisada puhastamise nupp, et kasutaja saaks genereerida mitu korda ilma vahepeal programmi sulgemata. Selleks oli vaja lisada iga genereerimise järel tühi rida eelmise tulemuse järele ning kuvada järgmine tulemus selle alla. Puhasta nupp pidi kustutama kogu teksti tulemuse väljal. Sellega kaasnes aga probleem, et programm ei salvestanud kasutaja algset sisendit, ent kasutamisel võib tekkida tekkida olukord, kus kasutaja ei taha lisada kokku 13 väljale uuesti kõiki andmeid, vaid tahab ainult paari parameetrit muuta - selleks kirjutasin juurde funktsionaalsuse, mis jätab meelde kasutaja sisendi ega kustuta seda, vaid hoiab seal seni, kuni kasutaja need andmed ise muudab või kustutab.

2.4.2 Graafiline disain

Graafiline disain vajas natukene rohkem aega, kui algselt oodatud. See sai loodud IntelliJ-sse sisse-ehitatud *Scene Builder* abil, mis kuvas mulle rakenduse visuaalse kuju ning sain seal muuta teksti asukohta, stiili ja sisu ilma seda käsitsi koodis muutmata ning ei pidanud iga muudatuse järel jälle programmi avama. Kahjuks aga ei olnud stseeni ehitajas kõike funktsionaalsust, mida oli vaja programmi võimalikult kasutajasõbralikuks disainimiseks. Seega asusin järgneva osa jaoks käsitsi koodi kirjutama.
Leidsin, et parim viis loogiliselt kasutajale teavet kuvada on jagada see osadeks: kõige üleval keskel sisestab kasutaja, millised kaardid on tema jaoks olulised; järgmises plokis on kõrvuti iga mängija kohtakolm sisendküsimust; seejärel on koht kasutajale programmi töö kohta olulise info kuvamiseks; selle all nupud genereerimiseks ja puhastamiseks ning viimaks tulemuste tekstikast. Kõige keerulisemaks kujunes mängijate kaartide info sektsioon, kus oli vaja kõrvuti kuvada kõik neli mängijat nende kohta teatava infoga. Leidsin, et kõige lihtsam lahendus on luua sellesse sektsiooni kuue rea ja nelja tulbaga tabel, kus iga tulp on mõeldud eraldi mängija info jaoks ning ridadel on vaheldumisi küsimused ja sisendväljad. Lisades tulba lahtritele ka vahed ning lubades ühes lahtris teksti mitmel real, sain soovitud tulemuse.

Samuti oli vaja ümber teha tulemuste lahter, et sealne tekst oleks kasutajale lihtsasti loetav. Ka selle parim lahendus tundus olevat luua tabel, kus on samuti neli tulpa mängijate jaoks ning kuni kaheksa rida potentsiaalseks väljundiks. Kaheksa rida selleks, et väljastada selle mängija jaoks kõik kaheksa tõenäosust maksimaalselt kolme kaardi puhul:

1) ei ole ühtegi kaarti
2) on ainult esimene
3) on ainult teine
4) on ainult kolmas
5) on esimene ja teine, aga pole kolmandat
6) on esimene ja kolmas, aga pole teist
7) on teine ja kolmas, aga pole esimest
8) on kõik kolm kaarti.

Viimaks oli vaja rakenduse aknale anda suuruse parameetrid klassikalise resolutsiooniga arvuti jaoks (1920*1080 pikslit ekraanile) selliselt, et info ära ei kaoks. Selleks kasutasin *fit* funktsionaalsust, mis loob akna täpselt sellises suuruses, et kõik info oleks kohe aknas näha. Sellega kaasnes aga probleem, et järjest tulemusi genereerides kadusid viimased tulemused ära, sest ekraaniruum sai otsa. Selle lahenduseks oli vaja lisada kerimisribad, mis lahendavad probleemi ka juhul, kui kasutaja ekraani resolutsioon erineb standardist. Selleks lisasin kogu visuaalse osa koodi *scrollbar* funktsionaalsusesse, et saaks programmi sisu kerida nii horisontaalselt kui vertikaalselt.

2.4.3 Programmi muutmine .exe failiks

Kõigest eelnevast aga ei piisa programmi jagamiseks bridži kogukonnas, sest hetkel eeldab programm Java programmeerimiskeele olemasolu arvutis ning kasutajalt teadmisi, kuidas programm koodi kaudu käivitada. Küsisin Eesti noortekoondise kolme treeneri käest, kuidas nemad tahaksid, et programm nendeni jõuaks ning sain kõigilt ühtse vastuse: programm peab olema nende arvutis lihtsalt topeltklõpsuga kävitatav. Selle parimaks lahenduseks tundus mulle muuta .jar fail .exe failiks, mille saan kokkupakitud kaustaga neile ja teistele soovijatele saata. Selle kohta ma kahjuks ei leidnud ühtegi IntelliJ sisseehitatud liidest, seega tuli lahendus leida mujalt.

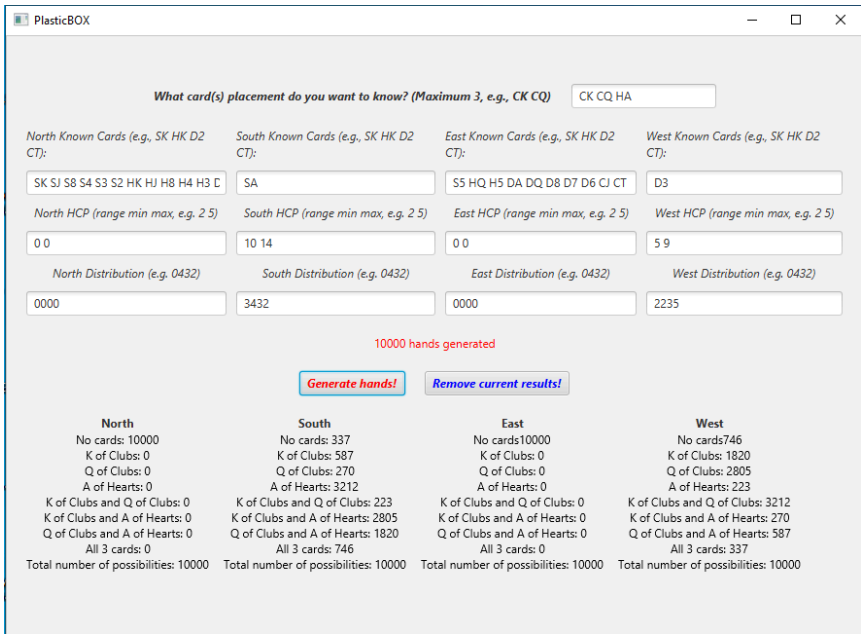
Leidsin programmi nimega launch4j [9], mis teisendab .jar faili .exe failiks. Programmiga tuli kaasa ka õpetus selle tegemiseks, mille järgimisel kujunes aga probleeme, kus programm ei käivitunud või konversioon ei olnud edukas vale Java SDK versiooni tõttu. Proovisin probleemi lahendada nii uuemaid kui vanemaid SDK versioone kasutades, kuid probleemi lahendada ma kahjuks ei suutnud. Küll aga meenus, et olen ühe varasema hobitöö puhul kokku puutunud sarnase olukorraga, kus väidetavalt on probleem tarkvara versioonides, aga tegelikult asub probleem hoopis mujal. Otsustasin luua täiesti eraldi uue klassi launcher.java, mis sisaldas endas täpselt ühte funktsionaalsust - käivitada programm, mis sisaldab endas tegelikku funktsionaalsust - ning proovida see muuta .exe programmiks. Selgus, et see oligi toimiv lahendus ning loodud .exe programm avaneb ja toimib igas arvutis, kus on olemas Java 21. Kahjuks ei leidnud ma head lahendust, kus kasutaja ei peaks ise Javat endale installeerima. Saaksin küll Java ka programmiga kaasa pakkida, aga see muudaks faili liiga suureks ning lihtsam lahendus tundus olevat anda kaasa juhend, kuidas endale korrektne Java installida.

3. Programmi kasutamine ja testimine

Järgnevas peatükis kirjeldatakse, kuidas kasutada programmi funktsionaalsust ja selle rakendamist bridžiõppel. Selgitatakse, kuidas programmi testiti ja selle tulemuste õigsust kontrolliti. Lisaks kirjeldatakse ka funktsionaalsust, mis on plaanis tulevikus lisada, kuid mille lisamine bakalaureuse töö raames oleks kujunenud liialt mahukaks.

3.1 Programmi kasutamine

Programmi käivitamisel näeb kasutaja enda ees minimalistliku disainiga akent 13 küsimusega, millele kasutaja vastama peab. Iga küsimuse juures on ka kirjas, millisel kujul vastust oodatakse. (Joonis 8) Saatsin kuvatõmmise programmi aknast ka kolmele bridžimängijale ning nad kinnitasid, et küsimused ja oodatav sisend on kasutaja jaoks loogiline. Sama ei öelnud aga kaks tuttavat, kes bridži ei mängi aga leidsin, et kuna programm on mõeldud juba edasijõudnud mängijatele, siis ei ole tarvis kasutajate sihtgruppi arvestades lisada rohkem õpetust programmi kasutamiseks vähemalt esialgu. Peale sisendi andmist programmile saab kasutaja näha genereeritud tulemusi ning teha enda jaoks vajalikud järeldused - samuti vajadusel sisendit muuta. Kasutajale ei kuvata praeguse programmi versiooni puhul programmi aknas genereeritud jaotusi vaid ainult lõpptulemust, kuid soovi korral pääseb kasutaja sellele ligi konsooli kaudu.



Joonis 8, Kuvatõmmis valminud programmist

3.2 Programmi testimine

Programmi testimise puhul oli vaja kinnitada kolm programmi olulist osa:

1. Programm tuvastab, kas antud sisend on vastavuses bridži reeglitega. Selle probleemi lahendamise käik on kirjeldatud peatükis 2.4.1. Testimise käigus testisin kõiki valesid sisestamisviise ning vastava sisuga teade kuvatakse programmi kasutajale.
2. Programm toimib erineva konfiguratsiooniga seadmetes edukalt ehk korrektse kiirusega ja programmi aken kuvatakse kasutajale korrektselt. Kokkupakitud faili jagasin iseendaga kokku kaheksas mulle kättesaadavas erinevas seadmes proovimiseks, nendeks olid: Windowsi virtuaalmasin MacOS ja Linux süsteemis; avamine ekraanidel resolutsioonidega 720p, 1080p, 1440p ja 2160p; avamine kümme aastat vanas sülearvutis ja võimsas mänguri arvutis, kus ajaline kulu erines keskmiselt ainult paar sekundit. Kõik testid olid edukad ning suuri muudatusi programmis tegema ei pidanud.
3. Kasutajale kuvatud tulemused on matemaatiliselt piisavalt täpsed. Selleks proovisin mitmel korral sama sisendiga genereerida 100, 1000, 10 000, 100 000 ja 1 000 000 jaotust ning leida, kas tekkinud erinevused on olulise suurusega. Testi tulemusena oli näha, et võrreldes 1 000 000 genereeritud jaotuse tulemusega erines 100 jaotus maksimaalselt kuni 20 protsendi võrra, 1000 puhul kuni kuus protsenti, 10 000 puhul kuni kaks protsenti ning 100 000 puhul kõigest 0,8%, kuid viimase puhul oli ajaline kulu juba liiga suur - enam kui kümme korda aeglasem kui 10 000 jaotuse puhul. Seega otsustasin 10 000 jaotuse genereerimise kasuks. Kuna programm genereerib jaotusi korrektselt bridži reeglite alusel, siis ei ole arvutuslik testimine tulemuste õigsuse kontrollimiseks vajalik, sest programm ei arvuta tõenäosusi vaid märgib iga genereeritud jaotuse puhul kaartide asukoha ning väljastab lõpuks kordade arvu, mil kaart või kaardid olid kindla mängija käes.

3.3 Tulevikus lisatav funktsionaalsus

Programmi soovin ma tulevikus täiendada veel väga mitme aspektiga kas magistritöö raames või kui programmi vastu tuntakse pärast esmase versiooni kasutamist suuremat huvi. Kuna bakalaureusetööl on kindlad mahu- ja ajapiirangud, on järgnevad funktsionaalsused programmi lisamata, kuid kirjeldan lühidalt nende sisu ja rakendamise ideed.

3.3.1 Mastipikkuste vahemike arvestamine

Hetkel arvestab programm ainult kindlate etteantud mastipikkustega. Küll aga oleks tulevikus oluline kasutajakogemuse mugavdamiseks lisada võimalus anda ette mastipikkuste vahemikud, kus on näiteks teada, et Northi mängijal on käes neli-viis pada, üks kuni kolm ärtut, kaks kuni neli ruutut ja kaks kuni neli ristit. Hetkel peab kasutaja iga mastijagumise sisestama eraldi. Seda funktsionaalsust pole hetkel lisatud ajalise keerukuse tõttu, sest programmi töö muutus seda kasutades liialt aeglaseks. Idee on lahendada probleem sarnaselt pildipunktide vahemikega, kus esmalt proovib programm leida vahemikest kõikvõimalikud kombinatsioonid mastipikkustele ning genereerida iga jaotust, valides suvalise sobiliku kombinatsiooni.

3.3.2 Kokkulepitud signaalid ja pakkumise stiil

Bridžilauas on mängijatel võimalik omavahel vahetada ka signaale, kus spetsiifilistes olukordades võib käidud kaart anda lisainformatsiooni. Näiteks kui lepinguks on kolm trumbita ning East teeb avakäiguks D3, siis võib West teada, et tema partneril on neli või viis ruutut käes, sest nad käivad avakäiguks tugevuselt alati neljanda kaardi. Samuti võib mängu käigus selguda informatsioon, et partneril peab kindlasti olema üks kõrgetest ärtu piltidest ehk äss või emand või kuningas. Hetkel programm sellega ei arvesta, sest see ei ole lihtne funktsionaalsus, mida programmile lisada. Plaan on tulevikus lisada võimalus kasutajal sisestada eeldusi mingite kindlate parameetrite järgi, millest programm saab aru. Näiteks klassifitseerida, mida tähendab suur kaart, keskmine kaart ja väike kaart ning kontrollida, kas genereeritud jaotus vastab ka nendele eeldustele.

Samuti on mängijatel omavahel paaris mängides tihti kokku lepitud kindel pakkumise stiil. Näiteks kui enamasti tähendab 2S avapakkumine, et mängijal on viis kuni kümme pildipunkti ja viis kuni seitse pada, siis on mängijaid, kellel on juures lisaeeldus, et sellise pakkumise

tegemiseks peab olema mängijal vähemalt kaks pildikaarti neljast või ei tohi olla üheski teises mastis ässa. Hetkel peab mõlemad järeldused tegema kasutaja ise ning lisama teadaolevate kaartide hulka ka kombinatsioone kaartidest, mille asukohta ta tegelikult kindlalt ei tea, aga oskab teha tugeva eelduse.

3.3.3 Välistavad eeldused

Tihti selgub pakkumise või mängu käigus mingeid välistavaid eeldusi. Näiteks kui West teeb avapakkumiseks 1NT, North passib, East pakub 3NT ning kõik kolm mängijat passivad, siis hetkel saab programmi jaoks ette anda eelduse, et Northil võivad käes olla ükskõik millised 13 kaarti. Kui aga minu partner oleks North, siis meie süsteemsete kokkulepete kohaselt saaksin ma enda partnerile lisada välistavad piirangud: partneril ei ole rohkem kui 15 punkti, ei ole kuue-kaardilist kallismasti, ei ole mõlemas kallismastis nelja kaarti, ei ole mõlemas odavmastis viite kaarti, ei ole samaaegselt viit kaarti ühes kallismastis ja neljakaarti ühes odavmastis. Hetkel ei saa programmile ette anda välistavaid eeldusi, kuid tulevikus on plaan need programmile anda sarnaselt signaali tähendustega, kus programm kontrollib pärast jaotuse genereerimist, et mõnel mängijal ei oleks käes sellised 13 kaarti, millega oleks ta süsteemselt olnud kohustatud pakkuma teisiti.

3.3.4 Visuaalsed uuendused

Kui lisada programmile kogu eespool kirjeldatud funktsionaalsus, peab kasutaja iga kord väga palju andmeid käsitsi sisendama ning see on ajakulukas. Programmile võiks lisada teadaolevate kaartide lisamise ühe klikiga kindlale mängijale - näiteks lisada ekraanile terve kaardipakk, kust mängija saab lohistada kaardid mängijatele, kelle käes vastav kaart kindlalt on. Samuti võiks pildipunktide vahemiku anda ette väärtuste ribana, millel saab vahemikku sättida. Lisana võiks programmil olla ka võimalus vajutada nuppu tulemuste kõrval “Vaata jaotusi” ning kasutajale avaneb uus aken, kus ta saab soovi korral uurida kõiki genereeritud jaotusi.

3.3.5 Telefoniäpi loomine

Kuna igal noorel on alati kaasas nutitelefon, oleks palju loogilisem seal kiirelt programm avada ning mõnda juhtunud olukorda analüüsida. Erinevalt treeneristest ei ole arvuti kasutamine turniiridel mängijatele tihti mugav ega praktiline, seega saavad praegust versiooni kasutada

peamiselt treenerid kas ise analüüsi koostamiseks või mängijate soovil neile soovitud olukorra analüüsi näitamiseks. Nutitelefoni äpi loomisel tundub aga esmapilgul olevat mitu probleemi, mille lahendusteni autor veel jõudnud ei ole: kuidas säilitada väiksema võimsusega seadmes analüüsi kiirus, kuidas anda paljude sisenditega programmile väiksel ekraanil lihtne visuaalne kuju, kuidas jagada programmi väljaspool Google Play jt poode. Küll aga on see oluline punkt, kui programmi tulevikus edasi arendada.

Kokkuvõte

Käesoleva töö eesmärgiks oli välja mõelda ja luua rakendus, mis aitaks bridžimängijatel ja -treeneritel paremini analüüsida mängus toimunut.

Töö käigus valmis Java programm, mis analüüsib mängijale teadaolevat informatsiooni ning annab vastuseks mängijat huvitavate kaartide asukohtade tõenäosused. Saadud info põhjal on bridžimängijal võimalik teha järeldused parema tulemuse saavutamiseks sarnases olukorras tulevikus.

Töös analüüsiti lähemalt selle sisulise programmi loomise vajadust bridži ja olemasolevate programmide kontekstis. Toodi välja koodi kirjutamise protsess: millised olid keerulised kohad, mida pidi arvesse võtma ning missuguseid võimalusi kasutati toimiva lahenduse loomiseks. Samuti kirjeldati programmi kättesaadavaks tegemist kasutajatele ja potentsiaalseid tulevasi edasiarendusi.

Valminud programmi loomisel kasutati erinevaid Tartu Ülikooli informaatika erialal õpitud teadmisi nagu näiteks Java kasutamine, graafilise disaini loomine ja kombinatoorika; tagasisidet juhendajatelt programmi ajalise keerukuse optimeerimiseks ning varasematest projektidest koodi kirjutamise kohta saadud oskusi. Töö eesmärk saavutati edukalt ning programmi kasutatab juba mitu Eesti bridžimängijat, kuid laiemaks kasutamiseks arendatakse programmi veel edasi.

Viidatud kirjandus

[1] World Bridge Federation, http://www.worldbridge.org/what-is-bridge/ 04.12.2023

[2] Funbridge, https://www.funbridge.com/counting-bridge 04.12.2023

[3] Edwin B. Kantar, Defensive bridge play complete, Ameerika Ühendriigid: Wilshire Book Company, 1974, 3

[4] Bogdan Vulcan, https://www.quora.com/What-is-the-more-decisive-factor-in-the-game-of-bridge-bidding-or-play 08.01.2024

[5] David Bird, Twelve important bridge lessons on defense, Suurbritannia: Master Point Press, 2023, 4-12

[6] Cuebids, https://cuebids.com/ 05.12.2023

[7] Bridgebase, https://www.bridgebase.com/ 05.12.2023

[8] Dealmaster, https://dealmaster.com/ 05.12.2023

[9] launch4j, https://launch4j.sourceforge.net/ 10.04.2024

Lisad

I Bridži reeglid

Maailma bridži föderatsiooni kinnitatud reeglid:
http://www.worldbridge.org/wp-content/uploads/2016/12/2017LawsofDuplicateBridge-paginated.pdf

II Githubi repositoorium

Valminud programm on leitav siit:

https://github.com/RikoPiirisild/bakalaureus2024piirisild.git

III Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Riko Piirisild**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose
   Bridži kaitsemängu analüüsi teostava treeningprogrammi loomine,

   mille juhendajad on **Ahti Põder** ja **Kristo Väljako**,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Riko Piirisild
23.04.2024